A New Algorithm for Standard Cell Global Routing

Jingsheng Cong[1]
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Bryan Preas
Xerox Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, California 94304

Abstract

In this paper, we present a new algorithm for standard cell global routing. The algorithm considers all of the interconnection nets in parallel; this produces superior results since information about all of the nets is available throughout the global routing process. We formulate the global routing as finding the optimal spanning forest (a generalization of optimal spanning trees) on a graph that contains all of the interconnection information. The results of several theorems allow us to prune many non-optimal connections before the process begins. This approach successfully solves the net ordering and congestion prediction problems which other approaches suffer. The new algorithm was implemented as part of the DATools system at Xerox PARC. The benchmarks from the Physical Design Workshop are used as part of the comparison suite. The new algorithm achieves up to 11% area reduction compared to the previous global routing package used in the DATools system and obtains up to 17% reduction in the total channel density compared to the Timberwolf 4.2 package. In no case does the new algorithm do worse than its competitors.

1. Introduction

The standard cell design style is widely used in the design of VLSI circuits; the cells (either obtained from a library or constructed) are arranged in horizontal rows. Due to the inherent complexity of physical design, the process is usually divided into three steps: placement, global routing and channel routing. During placement, we determine the row and the position within that row for each cell in the logic design. Next, during global routing, we determine the connection pattern, or topology, for each net. Global routing adds *feedthrough* cells to make connections across the cell rows, selects the pins (from the pins connected within the cells) to connect with wiring, and determines the routing channels in which the wiring should lie. Then, each channel is routed as a separate channel routing problem. A significant amount of work has been done on channel routing. There are several good channel routing algorithms which can consistently produce routing solutions that are no more than a few tracks above channel density. Also, modern placement programs can generate high quality placements for standard cell designs. However, we see only limited progress in standard cell global routing. Most global routing discussions are based on general cell design models.

The global router in the Highland system [Ro84] was used as the benchmark for standard cell global routing at the Physical Design Workshop on Placement and Floorplanning [Pr87]. It builds a minimum spanning tree for each net. The cost for each net edge is a function of channel density, feedthrough availability and wire length. An optimization step follows to try to improve the solution. Supowit [Su83] gives an *odd-even* heuristic algorithm for standard cell global routing. It produces a solution within a factor of 1.5 of the optimal solution, but his result applies only to problems in which all the nets have only two pins. Another global router developed for standard cell designs is part of the Timberwolf package [SeSa86]. It first connects each net by a minimum spanning tree based on wire lengths. Then it uses iteration (simulated annealing with t = 0) to improve the assignment of net segments to channels. The previous standard cell global router [Pr86] used in the DATools system at Xerox PARC [BaMS88] generates the minimum number of feedthroughs. Feedthroughs are added only to connect the nets. The feedthroughs are then placed, along with the other cells, as part of a row-based placement improvement step. More recent work on standard cell global routing was done by Mowchenko and Ma [MoMa87]; they generalized the left edge algorithm for channel routing to global routing. In [AoKu83] and [LiSS86], a similar problem is addressed for gate array designs, where the objective is to minimize the maximum channel density.

A careful study shows that these approaches face one or more of the following problems:

(1) The final solutions produced are sensitive to the order in which the nets are considered because the nets are connected one by one. However, little is known about what is a good net order. (In [MoMa87], they avoided net ordering problems by processing channel by channel. But still, it is difficult to choose a good channel order.)

(2) It is very difficult for these global routers to predict congested areas in channels when they add net segments. This is especially true in the early stages when most net segments have not been included.

(3) The net connection patterns that can be produced are restricted by the algorithm. Only a few predetermined topologies are allowed.

In this paper, we present a new algorithm for standard cell global routing which successfully overcomes the problems mentioned above. This algorithm processes all the nets in parallel, so the results are independent of the order in which the nets are considered. Furthermore, better results are produced since information about all of the nets is available throughout the global routing process. We introduce the net connection graph and formulate the problem as finding an optimal spanning forest of the net connection graph. According to a theorem in Section 3.2, we can simplify the net connection graph by pruning a large number of non-optimal connections. This enables our algorithm to consider all the possible optimal connections in all the channels. Thus, our algorithm can predict very accurately the densest areas in each channel and, therefore, distribute density evenly over all channels to minimize the *total channel density*. (Total channel density is the sum of the channel density over all of the channels.)

The remainder of this paper describes the algorithm. Section 2 defines the problem that is being solved and the two-stage algorithm is discussed in Section 3. The computational complexity of the algorithm is summarized in Section 4, and comparative results are presented in Section 5. The new algorithm gives 6% to 11% better results than the global routing algorithm that it replaced and 5% to 17% better results than that produced by the Timberwolf 4.2 global router.

2. Formulation of the Problem

The goal of global routing is to determine the connection pattern for each net and achieve the best utilization of chip area. The connection pattern is defined by positions for feedthroughs, the pins to be connected, and channels in which the net segments that connect the pins lie. *Chip area* is equal to the product of the width of the chip and the height of the chip, where the *width of the chip* is the maximum length (including any feedthroughs) of any row of the chip, and the *height of the chip* is the sum of the heights of all cell rows and the total channel density times the line-to-line spacing.

We define the *net connection graph* to be an undirected graph $G = (V, E)$, where V is the set of pins currently in the design. An edge (p_i, p_j) is in E if the two pins p_i and p_j are in the same net. We also call (p_i, p_j) a *net segment* if p_i and p_j are in the same channel. Clearly, each net in the net connection graph is a *connected component* and is represented by a complete graph on the pins of the net. Note that V may grow as we perform the global routing because new pins are introduced when feedthroughs are added. A *global routing solution* is a spanning forest of the net connection graph. A spanning forest which yields the minimum

[1] This work was performed when this author was with Xerox PARC during the summer of 1987.

chip area is called an *optimal spanning forest.*

There are two problems involved in standard cell global routing. The first problem is to determine whether and where to add feedthroughs. Generally speaking, feedthroughs have two functions. One function is to complete the connections among pins that make up the nets. For the example shown in Figure 2-1(a), we have to insert a feedthrough in row 2 to complete the connections for the net as shown in Figure 2-1(b). The other function of feedthroughs is to reduce the total channel density. Consider the net shown in Figure 2-2(a). Although we can complete the connection without adding any feedthroughs, by adding a feedthrough in row 2 we save a long wire in channel 2. This may reduce the total channel density. The second problem in standard cell global routing is to determine the net segments to complete the connection of the nets after feedthroughs have been added. Figure 2-3 shows two different choices of net segments to connect a net. At this stage, since the width of the chip is fixed, the problem is to build a spanning forest within the net connection graph to minimize the total channel density.

In [Su83] and [MoMa87], they assume that all the feedthroughs have been added and only address the second problem: determining the net segments. [SeSa86] and [Pr86] use a simple method to determine feedthrough locations to complete the connections. Then they concentrate on the second problem to determine the net segments.

Solutions to the global routing problem are very difficult; we can show the problem of finding an optimal spanning forest is NP-hard. In fact, the problem of determining net segments itself is already NP-hard (even assuming no feedthroughs need to be added). The NP-hardness result even holds for more restricted subcases. It is shown in [Ka86] that the problem of determining net segments is still NP-hard if each net has only two logical pins.

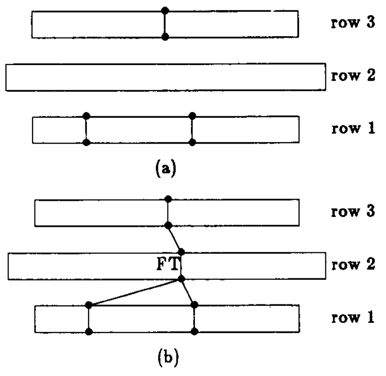

Figure 2-1 A feedthrough allows a net to cross a cell row. The feedthrough in row 2 is required to complete the connection.

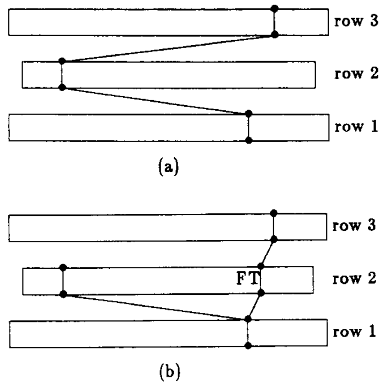

Figure 2-2 Feedthroughs can also be used to reduce the total channel density.

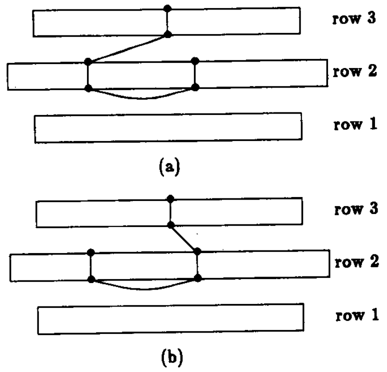

Figure 2-3 For the net shown, two different choices of net segments are available in the channel between rows 2 and 3.

We also show that the problem of determining net segments is NP-hard even if there are only two channels (see [CoPr88]).

In the rest of this paper, we present an efficient algorithm which constructs two minimum weighted spanning forests in two stages to approximate the optimal spanning forest.

3. The Standard Cell Global Routing Algorithm

Our global router works in two stages. In the first stage, we determine all the feedthroughs to be added and determine their locations within the rows. In the second stage, we choose which physical pins will be connected (and thus choose the net segments) to complete connections for each net.

3.1. Determine Feedthroughs

In most previous algorithms, feedthroughs are added only when connections for some nets cannot be completed. In our algorithm, we use feedthroughs not only to complete the connections but also to trade off the width and height of the chip. Additional feedthroughs can often reduce track density and thus reduce chip height with no expense in width. An additional feedthrough on other than the longest row will not increase the width of the chip.

We generalize Kruskal's minimum spanning tree algorithm [AhHU74] to build a minimum spanning forest of the net connection graph. Feedthroughs are determined by the intersections of cell rows and the edges in the minimum weighted spanning forest.

We weight each edge according to the length of the edge and the cost to insert feedthroughs for the edge. For each edge $e = (p_i, p_j)$ in the net connection graph connecting two pins p_i and p_j in the same net, we define the weight of e

$$w(e) = |x_i - x_j| + K \cdot \sum_{e \cap R_i \neq \varphi} weight(R_i)$$

where x_i and x_j are the horizontal coordinates for p_i and p_j, respectively. K is a constant factor. $e \cap R_i \neq \varphi$ means net edges e which intersect row R_i. $weight(R_i)$ is the weight of row R_i, which is based on the current length of row R_i. Assume we choose e to be included in the minimum weighted spanning forest. If p_i and p_j are in the same channel, we simply add e into the solution. If p_i and p_j are in different channels, we add feedthroughs $f_1, f_2, \cdots, f_l$ in rows $R_{i_1}, R_{i_2}, \cdots, R_{i_l}$ which intersect with e. Then we add the path from p_i to p_j through these feedthroughs into the solution. The cost of an edge thus defined is a function of both wire length and the cost of adding feedthroughs. By assigning different weights to different rows, we discourage adding feedthroughs on the longest rows since this will increase the width of the chip. On the other hand, when two pins are far apart we may connect them to nearby pins in the same net, even at the cost of extra feedthroughs. These extra feedthroughs may decrease the total channel density and thus decrease the height of the chip. We adjust the factor K to control the trade-off between the width and the height of a chip.

Note that the weight of each net edge is not static. After a feedthrough is added, some pin locations and the weights of some rows may change. If we construct the minimum spanning forest by building a minimum weighted spanning tree net by net, a different order for considering the nets will quite likely lead to a different result. However, there is no efficient algorithm available to determine an optimal net order. A key insight allows us to overcome this difficulty. Kruskal's minimum spanning tree algorithm [AhHU74] can be generalized to build the minimum spanning forest. Thus, instead of building minimum spanning trees net by net, our algorithm builds a minimum spanning forest directly by considering all of the nets in parallel as shown in Algorithm 3-1. In this algorithm, F represents the minimum spanning forest.

```
1. $E$ := all the edges in the net connection graph;
   $F := \varphi$;
2. while $E <> \varphi$ do
       remove the minimum weighted edge $e$ from $E$;
       if $F \bigcup \{e\}$ does not have a cycle
       then include $e$ (or the path induced by $e$) in $F$;
            update $E$ if feedthroughs are added;
   end-while
3. output $F$
```

Algorithm 3-1 Determine Feedthroughs (Stage 1 of Global Routing)

The advantages of this algorithm are clear. Since we consider the edges of all nets at the same time, the algorithm is independent of input net order. The spanning tree for each net gets an equal chance to grow. Information about all nets is available throughout the process.

It is necessary to show that this algorithm will converge since we keep adding new vertices (induced by feedthroughs) to the net connection graph. In [Ro84], a limit is set on the total number of vertices allowed for each net. However, for our algorithm we can show that $|V| - |F|$ decreases by one each time an edge (or a path) is added to F. (V is the set of vertices in the net connection graph and F is the set of edges in the partially constructed spanning forest.) Based on this observation, we have

Theorem 3-1 The Stage 1 algorithm will converge to a minimum spanning forest after $n - 1$ steps of inclusion, where n is the total number of pins originally in the design.

3.2. Determine Net Segments

After Stage 1, all of the required feedthroughs have been added and their positions have been determined; we will not add new feedthroughs. Thus, we can remove those edges that cross the cell rows (but are not *built-in* edges that represent feedthroughs or connections within the cells) from the net connection graph. The solution of Stage 2 is a spanning forest S within the net connection graph such that each edge in S lies entirely in one channel. Since the width of a chip is fixed after Stage 1, the objective in Stage 2 is to minimize the height of the chip by minimizing the total channel density.

Most previous approaches to global routing build a spanning tree for each net one by one. And for each net, these algorithms keep adding the minimum weighted feasible edge until a spanning tree is obtained. A serious problem exists with these approaches. It is very difficult to decide whether a net segment should be added to a channel since these algorithms have no knowledge of the density of the channel in the final solution, especially early in the execution of the algorithms when only few net segments are present. Also, these approaches face the problem of choosing a good order in which to process the nets.

To avoid these problems, we define a new algorithm. First, the basic approach is described; then we will show the performance improvements and refinements. The algorithm constructs a minimum weighted spanning forest to approximate the optimal spanning forest. We define the weight of an edge e to be $w(e) = d(e)/d$, where $d(e)$ is the maximum density over the edge in the channel to which e belongs, and d is the density of the channel. First, we put all the edges in the net connection graph into an edge set, S. Then we repeatedly remove the maximum weighted edge from S as long as we do not disconnect any net. We update the weights of edges in a channel whenever an edge is removed from that channel. The process terminates when S is a spanning forest. Clearly, this approach has two advantages:

(1) Since all the edges are considered from the start, the algorithm has global information and knows where the most congested areas are. The weight of each edge during the construction reflects the density over that edge in the resulting spanning forest;

(2) Since we process all nets in parallel, the result of our algorithm does not depend on the order in which nets are processed.

The following theorem shows that the result is close to the optimal spanning forest:

Theorem 3-2 Let S be a set of edges in the net connection graph. Let $C(S)$ be the subset of edges in S such that each edge in $C(S)$ lies on a cycle in S. The edge weights are equal to the density of the channel to which the edge belongs. We have

(1) If $C(S) \neq \varphi$, then there exists an edge e in $C(S)$ such that S contains an optimal spanning forest iff $S - \{e\}$ contains an optimal spanning forest;

(2) If $C(S) = \varphi$, then the optimal spanning forest based on S yields the same total channel density as S does.

For the proof, see [CoPr88]. This theorem justifies that there is a good chance the spanning forest constructed by our algorithm is an optimal spanning forest (remember that computing an optimal spanning forest deterministically is NP-hard).

We now describe the refinements to the basic algorithm. Remember there may be $\Theta(n^2)$ edges in the net connection graph at the beginning of Stage 2 (although we removed edges, other than the built-in edges or feedthroughs, which crossed cell rows), where n is the number of pins after Stage 1. A straightforward implementation of the above algorithm suffers two problems. First, since there are only $O(n)$ edges in the final spanning forest, we have to go through $\Theta(n^2)$ steps of edge deletion; this is quite time-consuming. Moreover, since most of the edges initially in S are to be removed, the weight of an edge at the beginning may not closely approximate the channel density over that edge in the final spanning forest. A careful study showed that we can overcome these two problems by simplifying the net connection graph before we compute the spanning forest S. We define the *simplified net connection graph SG* to be an undirected graph, whose vertex set is the set of pins in the design, and for two pins p_i and p_j of the same net, (p_i, p_j) is an edge of SG iff they satisfy one of the following conditions: (1) p_i and p_j are on the same cell or feedthrough and are connected internally; or (2) p_i and p_j are in the same channel and have no other pins of the same net between them. Figure 3-1 shows the example of a connected component induced by a net in the simplified net connection graph. Note it is much more sparse than a complete graph. In fact, we make the following claims:

Theorem 3-3 Let n and m be the number of vertices and edges, respectively, in the simplified net connection graph. Then

(1) $m \leq 1.5n$.

(2) The simplified net connection graph can be constructed in $O(n \log n)$ time, where n is the total number of pins in the design.

(3) The simplified net connection graph contains an optimal spanning forest.

For the proof, see [CoPr88]. The benefits from the theorem are clear. Since we only have to go through approximately $0.5n$ edge deletions, our algorithm runs much faster. Also, since only a relatively small number of edges in SG are to be removed, the weight of each edge more accurately measures the density over the edge in the resulted spanning forest. Moreover, we can compute the simplified net connection graph efficiently without losing the optimal spanning forest. We summarize our algorithm for Stage 2 as follows:

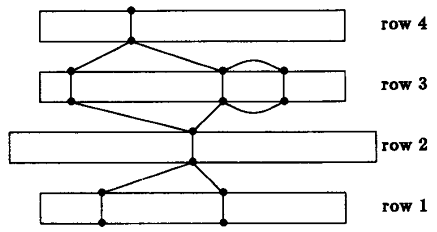

Figure 3-1 The connected component corresponding to a net in the simplified net connection graph is much more sparse than a complete graph. The optimal spanning forest is not lost by this simplification.

```
1. build the simplified net connection graph;
2. $S$ := all the edges in the simplified net connection graph;
3. repeat
       Remove the maximum weighted edge in $S$ on a cycle;
       Update edge weights for the affected edges;
   until $S$ is a spanning forest;
4. output $S$.
```

Algorithm 3-2 Determine the Net Segments (Stage 2 of Global Routing)

4. Complexity

Let n be the total number of physical pins originally in the design. (In modern standard cells, there are usually two physical pins per logical pin.) Let p be the average number of (physical) pins per net. Typical values for p range between 4 and 8. Based on Theorems 3-1 and 3-3, the overall time complexity of our new algorithm for standard cell global routing is $O(n^2 \max(p, \log n))$. For details of the analysis, see [CoPr88].

5. Experimental Results

We implemented our algorithm in the Cedar language running on Xerox Dorado workstations (2-MIPS machines) and incorporated it into the DATools system developed at Xerox PARC. Table 5-1 summarizes the examples used to compare the new algorithm with the previous global router in the DATools system and with the global router in Timberwolf 4.2. The counters and adders are the circuits synthesized by the DATools system when no performance requirements are imposed. Both types of circuits are simple, ripple-carry designs. Primary 1 and Primary 2 are the benchmarks from the Physical Design Workshop [Pr87].

Table 5-2 summarizes the experiments comparing the old and new algorithms. Compared to the previous standard cell global routing package used in the DATools system, the new algorithm achieves a 6% to 11% area reduction. In general, more feedthroughs are added by the new algorithm, but the chip widths are not increased. The extra feedthroughs are being used to reduce chip height by reducing total track density.

We also compared our global routing results with the results produced by the Timberwolf 4.2 global routing on the Physical Design Workshop benchmarks. In both examples, the global routing is performed on the same placement (the one produced by Timberwolf). Table 5-3 shows the comparisons on these examples. We obtained 5% to 17% reduction in total track density and over 20% reduction in the number of inserted feedthroughs compared to Timberwolf.

Ex.	# cells	# IOs	# nets	# pins
16-b adder	144	50	177	546
16-b ctr	173	56	206	609
32-b adder	288	98	355	1090
32-b ctr	342	104	396	1203
64-b adder	576	194	707	2178
64-b ctr	681	200	783	2393
Primary 1	752	81	904	2737
Primary 2	2907	107	3029	8758

Table 5-1 A summary of the example circuits used to compare the new algorithm with the previous algorithm and with the global router in Timberwolf.

Ex.	# of	previous algo.		new algo.		sav-
	rows	width	height	width	height	ing
16-b adder	4	1104	812	1104	764	6.3%
16-b ctr	5	1320	1120	1320	1008	11.1%
32-b adder	6	1528	1415	1528	1324	6.8%
32-b ctr	7	1904	1736	1904	1624	8.5%
64-b adder	8	2448	1956	2448	1860	5.1%
64-b ctr	9	3096	2744	3096	2456	11.7%

Table 5-2 Comparisons with the previous global routing package in the Xerox PARC DATools system.

Our global router is a straightforward implementation of the algorithm described here. As a result, there are a number of opportunities to significantly improve the results. Some standard cell global routing packages improve the global routing (and further reduce area) by exchanging adjacent cells in the same row or modifying the cell orientation ([Ro84, SeSa86, Pr86]). In addition, the cells in the Physical Design Workshop Benchmarks (Primary 1 and Primary 2) have a large number of built-in feedthroughs that are exploited by Timberwolf, but not by the global router described here. (The standard cells used at Xerox PARC do not have built-in feedthroughs so this feature was not included.) We did not implement these refinements in the current version of our global routing package.

Ex.	# of	Timberwolf		new algo.		improvement	
	rows	FTs	trks	FTs	trks	FTs	trks
P1	17	1380	223	1120	190	23.2%	17.4%
P2	29	4621	474	3761	449	22.9%	5.6%

Table 5-3 Comparisons with the global routing algorithm in Timberwolf 4.2 [SeSa86] on Primary 1 and Primary 2 benchmarks from the Physical Design Workshop.

6. Remarks and Conclusions

In this paper, we present a new algorithm for standard cell global routing. By processing all the nets in parallel, we avoid the problems associated with net ordering and the problems created by lack of information early in the global routing process. By simplifying the net connection graph and applying an iterative deletion algorithm for building spanning trees, we can more accurately predict congested areas. This global routing algorithm produces high quality solutions in polynomial time.

In both Stage 1 and Stage 2, the weights for net edges are dynamic. A significant amount of time is spent on updating edge weights in our current implementation. We are working on designing and implementing more efficient data structures to speed up the re-computation for edge weights.

References

[AhHU74] Aho, A., J. E. Hopcraft and J. D. Ullman, *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Co., Reading, MA, 1974.

[AoKu83] Aoshima, K. and E. S. Kuh, "Multi-channel optimization in gate-array LSI layout," *Proc. of IEEE International Symposium of Circuits and Systems* (1983) pp. 1005-1008.

[BaMS88] Barth, R., L. Monier, and B. Serlet, "PatchWork: layout from schematic notations," *Proc. of 25th Design Automation Conference* (1988) pp. 250-255.

[CoPr88] Cong, J. and B. Preas, "An improved approach for standard cell global routing", manuscript, 1988.

[Ka86] Kapoor, S., *Topics in the Design and Analysis of Combinatorial Algorithms*, Ph. D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, 1986.

[LiSS86] Lin, L., S. Sahni and E. Shragowitz, "An enhanced heuristic for multi-channel optimization in gate-array layout," *Proc. of ICCAD* (1986) pp. 242-244.

[MoMa87] Mowchenko, J. T. and C. S. R. Ma, "A new global routing algorithm for standard cell ICs," *Proc. of IEEE International Symposium on Circuits and Systems* (May, 1987) pp. 27-30.

[Pr86] Preas, B., "The Standard Cell Package," Xerox PARC internal document, 1986.

[Pr87] Preas, B., "Benchmarks for cell-based layout systems," *Proc. of 24th Design Automation Conference* (1987) pp. 319-320.

[Ro84] Roberts, K. A., "Automatic layout in the Highland system," *Proc. of ICCAD* (1984) pp. 224-226.

[SeSa86] Sechen, C. and A. Sangiovanni-Vincentelli, "Timberwolf3.2: a new standard cell placement and global routing package," *Proc. of 23rd Design Automation Conference* (1986) pp. 432-439.

[Su83] Supowit, K. J., "Reducing channel density in standard cell layout," *Proc. of 20th Design Automation Conference* (1983).